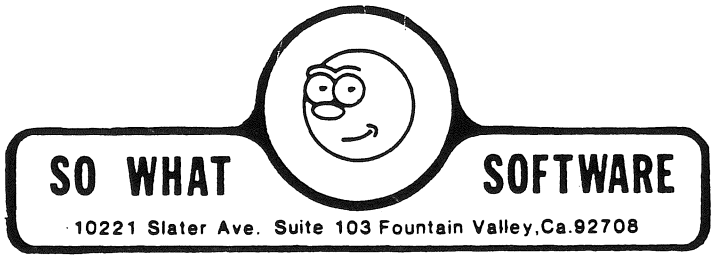

THE MENU MAKER

V 1.Ø

OVERVIEW

So What Software's Menu Maker combines three powerful binary files with an illustrative editor routine to make it easy to design custom "pull down" menus that respond to mouse action and key presses. Menus made with the Menu Maker have the look and feel of commercial application programs' menus, but they show exactly the options you want and run exactly the way you want them to!

Menu Maker menus can select and run any program that you can run from your IIgs' keyboard. Because the menus run under Applesoft BASIC, they are easily understood and modified by even beginning IIgs users. Just a little bit of Applesoft BASIC understanding is needed, but absolutely NO knowledge of machine language or Toolbox-oriented assemblers/compilers is required.

There are three menu programs on the /MENU.MKR disk: the Menu Maker program that you use in creating a custom menu, a demo program that shows some of the ways you can set up a menu, and a 'shell' that you can use as a starting point. The Menu Maker program is the one that comes up when you boot the /MENU.MKR disk. You can try it and the demo program out to see how they work, then LIST the Applesoft routines to see how they're written. Feel free to copy and use any of the Applesoft programming techniques we've used--we don't claim them to be either the best or the only way to go. Use whatever style you're comfortable with.

MENU SETUP

There are four steps in readying a menu:

1. Install the Menu Maker binary files (MENU.BIN.1, MENU.BIN.2, and MENU.BIN.3). From 'scratch', the three files are run in order; if you've got some of them already in memory, though, there are shortcuts. Look at the Menu Maker and Menu Demo program listings to see how to check if MENU.BIN.1 and/or MENU.BIN.3 are already present, and what to do if they are.

2. Install the Menu Maker Icon Sheet (the file MENU.ICONS) in an available memory bank and tell your menu where to find it. Look at the Startup program listing to see how this is done.

3. Set up the dimensions and fill in a two-dimensional string array (16x16 maximum size). Array entries are the text that will appear in the menu; each entry is identified for the array variable, A$(X,Y), where X denotes each item in the bar (the region across the top of the screen), and Y denotes each entry in a block (the region below a bar entry).

4. Initialize the menu by issuing the CALL MI command (from your menu program).

Of course, what your menu does when you click the mouse or press the keys is all up to you--that's taken care of by the menu's Applesoft BASIC routine.

Bar Text

Bar item text is identified in the first dimension of the array, in numerical order, without gaps.

```
Example:  A$(Ø,Ø) = "@"  (See Note, below.)
          A$(1,Ø) = "File"
          A$(2,Ø) = "Edit"
          A$(3,Ø) = "Quit"
           --- etc.---
```

Note: The "@" symbol entered at A$(Ø,Ø) yields a color-striped apple at the left of the menu bar. This is true for (Ø,Ø) only. The menu of the Menu Maker uses this feature.

The A$(X,Y) array should be dimensioned for the exact maximum number of entries for each dimension--dimensioning a larger one yields phantom entries that must be calculated when the menu parameters are generated. At best, the resulting menu has poor appearance; at worst, the menu may not be generated at all.

Block Text

Block entry text is identified in the second dimension of the array, in numerical order, without gaps. Block text entries are preceded by a Ø-5 code number to select text appearance in the finished menu.

```
Ø = Normal text
1 = De-emphasized text  ('gray-ed' out)
2 = Normal text w/ divider bar
3 = De-emphasized text w/ divider bar
4 = Normal text w/ check mark
5 = Normal text w/ check mark & divider bar
```

Special Menu Characters

Three special characters are available for block entries, the Open Apple, Closed Apple, and the alternate Open Apple character (the Funny-Looking Character, FLC). These characters are summoned by inserting a substitute character at the location you would want the special character to appear. The substitute characters are:

```
@ = Open Apple
^ = Closed Apple
~ = Alternate Open Apple (the FLC)
```

```
Example:  A$(6,1) = "4Plain        ^B"
          A$(6,2) = "ØBold         ^P"
          A$(6,3) = "2Italic       @I"
             --- etc.---
```

This example shows the Special Menu Characters used as modifiers to the keypress equivalents aligned at the right side of the menu block. Notice how every block item is preceded by a code number, and the array sequence is continuous (6,1; 6,2; etc). Leaving out a code number will cause an error message to appear, while having a gap in the array numbering will cause the menu block to end at the point where the inconsistency starts.

OPERATION

Three Calls and four data locations yield complete menu control:

1.	MI - Initialize.	$9400	37888
2.	MR - Reenter	$9403	37891
3.	MC - Close	$9406	37894
4.	MD - Menu Data Byte	$940F	37903
5.	BC - Text/Border Color	$940B	37899
6.	KC - Menu Background Color	$9409	37897
7.	VR - Array Variable	$940D	37901

Typical operation from within a program is to test that the cursor is within the active bar region and either the mouse button or a desired key combination has been pressed. Satisfaction of these two conditions when the mouse button has been clicked allows the Reentry Call (CALL MR), at which point the MENU.BIN.3 routine takes over and keeps control until the mouse button is released. Key presses can be noticed and acted on independently. Release of the mouse button while the cursor is in an active menu region (ie, a bar area or a block area) returns the Menu Data Byte, MD, identifying the menu item selected. Your menu program then does whatever you want it to do.

Data bytes BC and KC let you set the colors of the text/borders and the menu background, respectively. The color palette used by these values is the one used by the Icon Sheet installed when the menu is first installed. You can Poke different color values in for KC and BC--for best results, use these 16 values: Ø,17,34,...,238,255. You can use any IIgs paint program to change the color palette of the MENU.ICONS Icon Sheet, but don't change any of the sheet that shows the text and special characters!!

Data byte VR lets you change the 'active' array variable (eg, the 'A$' used in the discussions above). This means that you can have two (or more) separate menus available, just a mouse click or keypress away. The Menu Maker menu illustrates the technique. (There's more discussion of the VR variable in the Changing to Another Menu section.)

STORING VARIABLES

The Menu Maker program has an option that lets you store all the variables of a program in a special Applesoft filetype, a 'VARS' file. This method of handling variables eliminates the need for your program to 'declare' all the variable names at the beginning (which helps keep the program shorter), and it lets you set up variables to their former values almost instantly. Use the Menu Maker's 'Store Vars' option while you're creating a custom menu, if you wish.

The command to store variables is STORE VARS.FILE (or any filename you like) from the keyboard, or PRINT CHR$(4); "STORE VARS.FILE" from a program. Similarly, one gets variables back by a RESTORE VARS.FILE or PRINT CHR$(4); "RESTORE VARS.FILE" command. Check the Menu.Maker program listing to see how it's done.

We recommend that you save a copy of your menu program in a 'normal' format (ie, with all the variables--especially the array variables--shown in the listing). That's because the 'VARS' file format doesn't allow you to see on the screen what the variables are; if you want to change menu text, for instance, it'll be much easier to do it in the 'normal' way. Just use the 'VARS' file approach when you've got your menu text entries all laid out exactly as you want them.

SAMPLE PROGRAM LINES (After the menu is set up and ready to run.)

Standard Mouse/Cursor/Keypress Loop

```
10 CALL MS: CALL CO
20 CALL MS: CALL CO: IF PEEK (MB) = 255 THEN 100
30 K = PEEK (49152) : MK = PEEK (49189) : IF K < 128 THEN 20

    { K and MK are variables that designate what keyboard keys have been
      pressed; K is for 'regular' keys, while MK is for the 'modifier' keys,
      such as the Control, Shift, and Apple keys.  The Menu Maker shows you
      the K and MK values for any combination; use the info when you're
      designing your menu's logic. }

40 POKE 49168, 0
50 IF K = { a value between 128 and 255 } THEN 200
60 IF MK = { a value greater than 0 } THEN 300
70 GOTO 20    { No choice made, so go to beginning. }
100 IF PEEK (MV) >13 THEN 20  { The mouse button was pressed--was the cursor i
     the menu bar?  If not, go back to the beginning.}

110 CALL MR : X = PEEK (MD)  { Variable X tells which menu choice was made; it
     is decoded as shown in Lines 110, 120, and 130. }

120  B = INT (X/16)  { Bar item number }
130  M = X - INT (X/16) * 16  { Block entry number }
140  IF X = 255 THEN... { No selection was made, so... }
150  IF B = 1 AND M = 1 THEN...{A typical combo; go do something }
160  IF B = 1 AND M = 2 THEN...{ Another typical combo }
199  GOTO 20 { Gotta close this loop! }

200  { Do something because a certain key was pressed. }
300  { Do something because a 'modifier key' was pressed. }
399 GOTO 20 { End of the loop...go back to the beginning. }
```

The MENU.BIN.3 routine is shut down by the MC Call, at which time the menu disappears. To reacquire the menu, the MI Call must be issued; using the MR Call by mistake will cause unpredictable results.

With only one exception, bar or block text entry changes during program execution must be followed by the MI Call to recalculate and reformat the menu; otherwise, text may be plotted in undefined menu areas. (Since the MI Call generates the menu in a few hundredths of a second, this poses no practical limitation on speed.) The only exception is that the 0-9 code selecting block text status (eg, normal, de-emphasized, w/ or w/o check mark) may be changed "on the fly."

DISTRIBUTION TECHNIQUES

Here are three methods that you can use to deal with the returned data byte value after a menu selection has been made. The following sample code assumes a menu array of A$(2,2), while variable X equals the raw value returned in the menu data byte, M equals the menu bar item, and B equals the block entry.

TYPE 1: 2-PART DISTRIBUTOR

```
10   ON M GOTO 200,300

100  ON B GOTO 120,130
110  GOTO (routine 0,0)
120  GOTO (routine 0,1)
130  GOTO (routine 0,2)
200  ON B GOTO 220,230
210  GOTO (routine 1,0)
220  GOTO (routine 1,1)
230  GOTO (routine 1,2)
300  ON B GOTO 320,330
310  GOTO (routine 2,0)
320  GOTO (routine 2,1)
330  GOTO (routine 2,2)
```

TYPE 2: 2-FACTOR DISTRIBUTOR

```
100  IF M = 0 AND B = 0 THEN GOTO (routine 0,0)
110  IF M = 0 AND B = 1 THEN GOTO (routine 0,1)
120  IF M = 0 AND B = 2 THEN GOTO (routine 0,2)
130  IF M = 1 AND B = 0 THEN GOTO (routine 1,0)
140  IF M = 1 AND B = 1 THEN GOTO (routine 1,1)
150  IF M = 1 AND B = 2 THEN GOTO (routine 1,2)
160  IF M = 2 AND B = 0 THEN GOTO (routine 2,0)
170  IF M = 2 AND B = 1 THEN GOTO (routine 2,1)
180  IF M = 2 AND B = 2 THEN GOTO (routine 2,2)
```

TYPE 3: 1-FACTOR DISTRIBUTOR

```
100  IF X = 0 THEN GOTO (routine 0,0)
110  IF X = 1 THEN GOTO (routine 0,1)
120  IF X = 2 THEN GOTO (routine 0,2)
130  IF X = 16 THEN GOTO (routine 1,0)
140  IF X = 17 THEN GOTO (routine 1,1)
150  IF X = 18 THEN GOTO (routine 1,2)
160  IF X = 32 THEN GOTO (routine 2,0)
170  IF X = 33 THEN GOTO (routine 2,1)
180  IF X = 34 THEN GOTO (routine 2,2)
```

We find Type 1 to be the fastest of the three, as well as being the most compact. Type 2 is easier to trace through, but it takes up the most room. Type 3 is a more compact version of Type 2, using the raw data byte instead. You can LIST the Menu Maker program to get an idea of how these look in operation.

READING & WRITING THE BLOCK ITEM CODE NUMBER

There are times when you want a menu to show special notation about the status of a block item. This is usually gray-ing out an item or putting a check mark by it. The 'string slice' technique is used in order to read or write these code numbers; examples follow. (See the Block Text section, too.)

X = Bar item number Y = Block entry number N1 and N2 = Your numbers

Reading a Code Number

```
N1 = VAL (LEFT$ (A$ (X,Y), 1))
```

Writing a Code Number

```
LEFT$ (A$ (X,Y), 1) = STR$ (N2)
```

The Menu Maker program illustrates the technique. Select Item 4, Entry 4 several times to see the check mark coming and going.

RUNNING PROGRAMS

The preceding discussion deals with how to get to a 'Now What' point in your menu. Once you're there, it's up to you to tell your IIgs what to do next. Almost everyone uses a menu to run programs, so all you need to do is have the appropriate line of your menu issue the correct command to run the program of your choice. Here's an example:

```
5ØØ IF B = 1 AND M = 3 THEN PRINT CHR$(4);"PREFIX /APPLEWORKS" :
    PRINT CHR$(4); "RUN APLWORKS.SYSTEM"
```

This line sets the prefix to a disk having Appleworks on it and runs the system file. Any application program you can run by typing in commands from your keyboard can be run from your custom menu. All it takes is a little understanding of the requirements of your particular application and the issuing of 'run' commands via Applesoft.

ERROR MESSAGES

There are 6 error messages generated by the MENU.BIN.3 file. After the message is printed on the screen, the computer is put in Applesoft immediate mode so you can re-edit your program to eliminate the error.

MBIN.1 Not Installed: The menu needs MENU.BIN.1 to draw on the graphic screen. This message comes up at install time only.

Menu Array Not Dimensioned: A menu requires that a menu array (default A$) be dimensioned with a DIM statement prior to CALL-ing MI.

Menu Field Too Large: There are more than 15 block items specified.

Too Many Bar Selections: There are more than 16 bar items specified.

Bar Selections Too Wide: There are more than 39 characters specified in the menu bar (including auto-generated spaces between bar items)

Block Item Missing Code Number: Just what it says! (See Block Text section.)

CHANGING TO ANOTHER MENU

You can have more than one menu available, but it requires the changing of the array variable, VR, during your program. Here's how it's done:

An Applesoft variable name is stored in memory as a 2-byte (2-POKE) value. For a string array variable, the first byte is 'positive', while the second byte is 'negative'. Positive and negative in this case mean either greater than or less than 128. (Ø-127 are 'positive'; 128-255 are 'negative'.)

If the variable name has only one character, the first byte is the positive ASCII value for that character, and the second byte is 128 ($8Ø in hex).

Example: A$ = 65,128 ($41,$8Ø)

If the variable name has two characters, the first value is positive, and the second value is negative.

Example: AA$ = 65,193 ($41,$C1)

{{There are dozens of reference manuals that lay out all 255 ASCII values--consult the one you like best.}}

To have two menus available, all you do is dimension and fill in the different menu arrays, named as you see fit. POKE in your first array variable name at VR and VR+1, CALL MI, and then go to the primary program loop. When you want to use another menu, just CALL MC to shut down the current menu, POKE VR and VR+1 with the new menu array name values and CALL MI. Presto! You're off and running with the new menu. The Menu Maker program shows this in action; select the 'Other Menu' option to see it.

PITFALLS, GOTCHAs, AND OTHER STUFF

What follows is an assortment of things you ought to know and think about as you're creating and using custom menus. The list by no means covers all possibilities, so we urge you to examine the programs on the /MENU.MKR disk, read up a little on Applesoft BASIC programming (remember, Apple included a brief Applesoft tutorial manual in the box with your IIgs), and experiment.

1. The MENU.BIN files require some amount of free memory space to operate; for instance, calculating the menu layout is no simple task. Try to keep your menu program as short as you can--shoot for less than 5000 bytes. Use of the variables file technique helps cut down on program length.

2. Array variables are notorious memory hogs! If you're going to use the 'other menu' technique, keep the two arrays pretty small. If you don't, you'll get disappointing results, usually program crashes.

3. If you use a program on the /MENU.MKR disk as a starting point for a custom menu, delete all the 'useless' program lines, such as the REM statements.

4. Remember that all three of the MENU.BIN files have to be loaded in the correct sequence for the graphics and menu tasks to be carried out. If your menu program keeps crashing on startup, check this point. Also, if you use a separate Startup program (as we do with the Menu Maker), you need to restore the variables from the GRAPH.VARS file. The Menu Demo program has about the most compact way of getting started, restoring both graphics and menu variables from a single variables file. This method gives you the most room for your Applesoft menu routine, and we recommend you emulate it.

5. Start small and work up. The Menu Maker helps you get the 'menu' part of your program going, but it doesn't write the part that takes action when you've made a menu choice. Start with one or two options and add to your program as you become more confident.

6. The Menu Maker is one of several items in So What Software's 'Plain Brown Wrapper' series; others include the Screen Thief (for capturing any Super Hi-Res screen as a graphics file) and the ICONIX demo disk (which shows off much of the graphics power of our program ICONIX for the Apple IIgs). Ask for more info.

7. Speaking of ICONIX, if you'd like to add even more snap to your menus, you'll want to add ICONIX to your IIgs software library. The Menu Maker programs use a few of the many Applesoft BASIC commands that ICONIX provides--there's much, much more. ICONIX, SONIX, and DISC COMMANDER are our formal Apple IIgs programs (ie, with extensive features and full documentation...and vastly superior to 'plain brown wrapper' software such as the Menu Maker). They give the IIgs owner full control of his/her machine's outstanding graphics, sound, and file handling power. Take a look at the color info sheet enclosed for details on these three programs.

8. We're always interested in your comments and suggestions. Feel free to write us at 10221 Slater Ave., Suite 103, Fountain Valley, CA 92708.